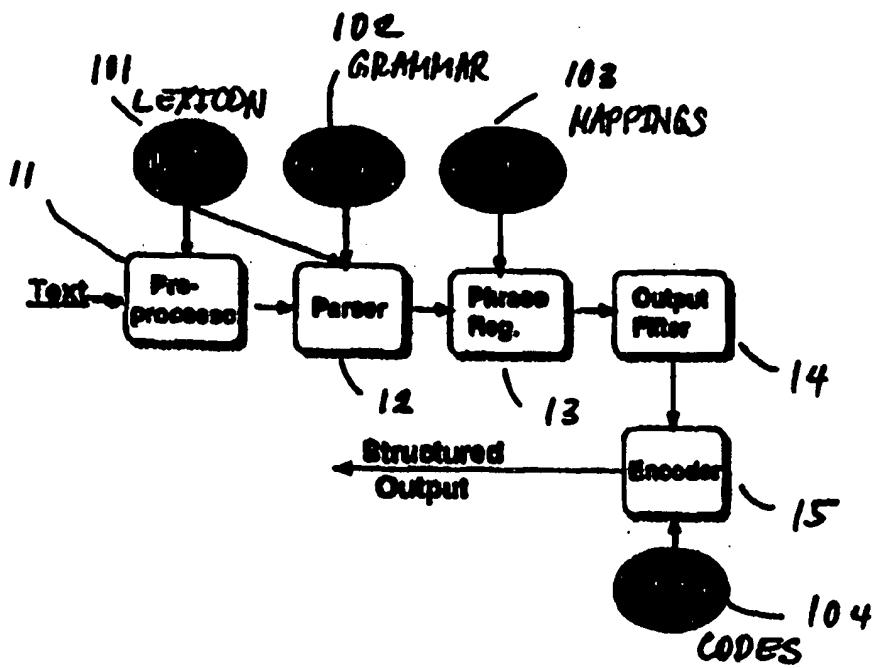




INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 17/22		A1	(11) International Publication Number: WO 98/19253 (43) International Publication Date: 7 May 1998 (07.05.98)
(21) International Application Number: PCT/US97/19362 (22) International Filing Date: 28 October 1997 (28.10.97)		(81) Designated States: CA, JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).	
(30) Priority Data: 08/738,889 28 October 1996 (28.10.96) US		Published <i>With international search report. Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i>	
(71) Applicant: TRUSTEES OF COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK [US/US]; 116th Street and Broadway, New York, NY 10027 (US).			
(72) Inventor: FRIEDMAN, Carol; 14 Dimitri Place, Larchmont, NY 10538 (US).			
(74) Agents: TANG, Henry et al.; Brumbaugh, Graves, Donohue & Raymond, 30 Rockefeller Plaza, New York, NY 10112-0228 (US).			

(54) Title: SYSTEM AND METHOD FOR MEDICAL LANGUAGE EXTRACTION AND ENCODING



(57) Abstract

In computerized processing of natural-language medical/clinical data including phrase parsing and regularizing, parameters are referred to whose values can be specified by the user. Thus, a computerized system can be provided with versatility, for the processing of the data originating in diverse domains, for example. Further to a parser (12) and a regularizer (13), the system includes a preprocessor (11), output filters (14) and an encoding mechanism (15).

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

Description

**System and Method for
Medical Language Extraction and Encoding**

5

Background of the Invention

This invention relates to natural language processing and, more specifically, to computerized processing of natural-language phrases found in medical/clinical data.

10 Clinical information as expressed by health care personnel is typically provided in natural language, e.g., in English. But, while phrases in natural language are convenient in interpersonal communication, the same typically does not apply to computerized applications such as automated quality assurance, clinical decision support, patient management, outcome studies, administration, research and literature 15 searching. Even where clinical data is available in electronic or computer-readable form, the data may remain inaccessible to computerized systems because of its form as narrative text.

For computerized applications, methods and systems have been developed for producing standardized, encoded representations of clinical information 20 from natural-language sources such as findings from examinations, medical history, progress notes, and discharge summaries. Special-purpose techniques have been used in different domains, e.g., general and specialized pathology, radiology, and surgery discharge reports.

Of particular further interest is a general approach which is based on 25 concepts and techniques described in the following papers:

C. Friedman et al., "A Conceptual Model for Clinical Radiology Reports". In: C. Safran, ed., Seventeenth Symposium for Computer Applications in Medical Care, New York, McGraw-Hill, March 1994, pp. 829-833;

30 C. Friedman et al., "A General Natural-Language Text Processor for Clinical Radiology", Journal of the American Medical Informatics Association, Vol. 1 (April 1994), pp. 161-174;

- 2 -

C. Friedman et al., "A Schema for Representing Medical Language Applied to Clinical Radiology", Journal of the American Medical Informatics Association, Vol. 1 (June 1994), pp. 233-248;

5 C. Friedman et al., "Natural Language Processing in an Operational Clinical Information System", Natural Language Engineering, Vol. 1 (March 1995), pp. 83-106.

Summary of the Invention

10 A preferred method for computerized processing of natural-language medical/clinical data includes basic steps here designated as phrase parsing and regularizing and, optionally, code selection. Further included, preferably, is a step of pre-processing prior to phrase parsing, and a step of output filtering. Output can be generated in the form of a printout, as a monitor display, as a database entry, or via 15 the "information highway", for example.

In processing, one or several parameters are referred to. The parameters are associated with options. To choose an option, the appropriate value is assigned to the parameter. A parameter can have a value by default. Of particular importance is the inclusion of a parameter which is associated with the 20 medical/clinical domain or subfield of the input data. Other parameters may be associated with the level of parsing accuracy desired, whether code selection is desired, the type of filtering, or the format of the output.

The method can be expressed in a high-level computer language such as Prolog, for example, for execution as a system on a suitable general-purpose 25 computer. In the following, the method and the system will be referred to by the acronym MedLEE, short for Medical Language Extraction and Encoding.

Brief Description of the Drawing

30 Fig. 1 is a diagram of the MedLEE system or "server".

- 3 -

Fig. 2 is a diagram of a system or application which has an interface for MedLee.

An Appendix hereto includes a printout of computer source code for a portion of MedLEE.

5

Detailed Description of Preferred Embodiments

A natural-language phrase included in medical/clinical data is understood as a delimited string comprising natural-language terms or words. The 10 string is computer-readable as obtained, e.g., from a pre-existing database, or from keyboard input, optical scanning of typed or handwritten text, or processed voice input. The delimiter may be a period, a semicolon, an end-of-message signal, a new-paragraph signal, or any other suitable symbol recognizable for this purpose. Within the phrase, the terms are separated by another delimiter, e.g., a blank or another 15 suitable symbol.

As a result of phrase parsing, terms in a natural-language phrase are classified, e.g., as referring to a body part, a body location, a clinical condition or a degree of certainty of a clinical condition, and the relationships between the terms are established and represented in a standard form. For example, in the phrase "moderate 20 cardiac enlargement", "moderate" is related to "enlargement" and cardiac is also related to "enlargement".

In the interest of versatility and applicability of the system to different domains, parsing is domain specific as a function of the value assigned to a parameter which the system refers to in parsing. Depending on the value of the domain 25 parameter, the appropriate rules can be referred to in parsing by the system.

While parsing may be based primarily on semantics or meaning, use of syntactic or grammatical information is not precluded.

Regularizing involves bringing together terms which may be discontiguous in a natural-language phrase but which belong together conceptually.

30 Regular forms or composites are obtained. Regularizing may involve reference to a separate knowledge base. For example, from each of the phrases "heart is enlarged",

- 4 -

"enlarged heart", "heart shows enlargement" and "cardiac enlargement", a regularizer can generate "enlarged heart".

In code selection, which is optional, a common, unique vocabulary term or code is assigned to each regular term by reference to yet another knowledge 5 base which may also be chosen domain specific. For example, in the domain of X-ray diagnostics, the term "cystic disease" has a different meaning as compared with the domain of mammography.

Fig. 1 shows a preprocessor module 11 by which natural-language input text is received. The preprocessor uses the lexicon knowledge base 101 and 10 handles abbreviations, which may be domain dependent. With the domain parameter properly set, the preprocessor refers to the proper knowledge base. For example, depending on the domain, the abbreviation "P.E." can be understood as physical examination or as pleural effusion. Also, the preprocessor determines phrase or sentence boundaries, and generates a list form for each phrase for further processing 15 by the parser module 12.

The parser module 12 also uses the lexicon 101, and a grammar module 102 to generate intermediate target forms. Thus, in addition to parsing of complete phrases, subphrase parsing can be used to advantage where highest accuracy is not required. In case a phrase cannot be parsed in its entirety, one or several 20 attempts can be made to parse a portion of the phrase for obtaining useful information in spite of some uncertainty. For example, subphrase parsing can be used in surveying discharge summaries.

With the parsed forms as input, and using mapping information 103, the phrase regularizer 13 composes regular terms as described above.

25 From the regularized phrases, the filter module 14 deletes information on the basis of parameter settings. For example, a parameter can be set to call for removal of negative findings.

The encoder module 15 uses a table of codes 104 to translate the regularized forms into unique concepts which are compatible with a clinical 30 controlled vocabulary.

- 5 -

Fig. 2 shows an interface module 21, and the MedLEE system 22 of Fig. 1. The interface module 21 may be domain-specific, and it may serve, e.g., to separate formatted sections from non-formatted sections in a report. Also, the interface 22 may serve to pass chosen parameter values to the MedLEE system 22 and 5 to pass output from the MedLEE system. For example, such an interface can be designed for communication over the World-Wide Web or a local network, for input to or output from MedLEE.

Conveniently, each module is software-implemented and stored in random-access memory of a suitable computer, e.g., a work-station computer. The 10 software can be in the form of executable object code, obtained, e.g., by compiling from source code. Source code interpretation is not precluded. Source code can be in the form of sequence-controlled instructions as in Fortran, Pascal or "C", for example. Alternatively, a rule-based system can be used such a Prolog, where suitable sequencing is chosen by the system at run-time.

15 An illustrative portion of the MedLEE system is shown in the Appendix in the form of a Prolog source listing with comments. The following is further to the comments.

Process_sents with get_inputsents, process_sects and outputresults reads in an input stream, processes sections of the input stream according to parameter 20 settings, and produces output according to the settings. Among parameters supplied to Process_sents are the following: Exam (specifying the domain), Mode (specifying the parsing mode), Amount (specifying the type of filtering), Type (specifying the output format) and Protocol (html or plain). Process_sents is called by another predicate, after user-specified parameters have been processed.

25 Process_sects with get_section and parse_sentences gets each section and generates intermediate output for the sentences in each section.

Outputresults with removefromtarg, write, writelines, markupsents and outputhl7 filters output if appropriate, produces output in the appropriate format and 30 optionally including formats tags for selected words of the original sentence, and produces error messages and an end-of-output message.

- 6 -

Setargs sets arguments or parameter values based on user input or by default.

Removefromtarg filters formatted output by leaving only positive clinical information and by removing negative findings from the formatted output.

5 Another parameter, parc, removes findings associated with past information from the formatted output. Any number of different filters can be included as suitable.

Writelines produces one line per finding, in list format.

Writeindentform and writeindentform2 produce output in indented form.

Markupsents envelopes the original sentence with tags so that the 10 clinical information is highlighted. Different types of information can be highlighted in different colors by use of an appropriate browser program such as Netscape, for example.

Outputhl7 converts output to appropriate form for database (xformtodb) and writes out the form in hl7 in coded format. This process uses 15 synonym knowledge and an encoding knowledge base.

- 7 -

APPENDIX

-8-

```

% Fall an unknown predicate
:- unknown(_,fail).

:- op(900, fy, [ \+, not, once ]).  % same priority and type as \+
:- op(700, xfx, [ \=, =- ] ).        % same priority and type as = or ==
:- dynamic(domain/1).             % domain being processed
:- dynamic(outputform/1).          % form of output (needed to distinguish
                                   % markup of text from formatting forms
:- dynamic(currentsect/1).        % section for outputting results
% Use when testing grammar - use default parameters for section,type,mode
testing (Exam,Form,Amount,Infile,Outfile) :-  

  see (Infile), seen, see (Infile),
  get_inputsents ([],Toklist),
  retractall (domain(_)),
  assert (domain (Exam)),
  retractall (paragno(_)),
  assert (paragno (1)),
  retractall (sentno(_)),
  assert (sentno (0)),
  retractall (outputform(_)),
  assert (outputform (Form)),
  retractall (currentsect (_)),
  process_testing (Toklist,Exam,Amount,Outfile).

process_testing ([],_,_,_).

process_testing (Toklist,Exam,Amount,Outfile) :-  

  get_sentence (Toklist, Sent, Rest),
  parse_sentences (Sent, Pmtlist, Outfail, Outndefs, Outunsents,
  testing, bp, Exam, _, _, 0),
  tell (Outfile),
  write_sentences (sent),
  outputform (Type),
  outputresults (Fmtlist, Outfail, Outfile, Outndefs, Outunsents,
  Outfile, Amount, Type, Exam, 1, 0, "exe,plain"),
  process_testing (Rest, Exam, Amount, Outfile).

```

```

process_sents(Exam, Sect, Mode, _, Outfile, Amount, Type, Campno, DocComp, Errfile,
  Unfile, Caller, Protocol) :-
  get_inputs([ ], Toklist), !, & read in and tokenize input
  tell(Outfile),
  ( % write out output label if writing using client/server model
    % or if protocol is html
    (Caller = server, !; Protocol = html, !), & beginning of output
    write('<html> <head> <title>'),
    write('Output Generated by MedLEE'),
    write('</title> </head>'). new_line(html),
    write('<body> <H1> Output Generated by MedLEE </H1> <HR> '),
    nl,
    new_line(html),
    nl
  ),
  (Toklist = [ ], !, !,
   write_message(Errfile, 'Input Error: No input entered', Caller,
   '<HR><H3> Application Message: No input entered </H3><HR>')
  ),
  !, % check for legitimate section header if Sect = 'mixed'
  Sect = mixed, get_section(Toklist, _, _, S, _), S = [ ], %no header found
  write_message(Errfile, 'Input Error: No legitimate section header',
  Caller,
  '<HR><H3> Application Message: Legitimate section header is missing </H
  3><HR>')
  ),
  retractall(domain(_)),
  assert(domain(Exam)), % used in grammar to permit certain rules
  retractall(outputform(_)),
  assert(outputform(Type)), % used in parser and formatter
  retractall(currentsect(_)), % used in formatting output
  assert(currentsect(Bact)).

```

```

markup(Toklist,Toklist2,Sect), % markup up in certain conditions
process_sects([Toklist2,Sect,Mode,Exam,Fmtlist,Failed,Undef,
  UnSent,1],
  outputresults(Fmtlist,Failed,Errfile,Undef,Unfile,UnSent,Outfile,
  Amount,Type,Exam,Compho,DocComp,_,Caller,Protocol)
).

% case where there is no more input.
process_sects([],_,_,_,_,_,_,_,_,_).

% sections are specified in input. there are several sections in input
process_sects([Toklist,mixed,Mode,Exam,Fmtlists,Fails,Undefs,
  UnSents,_]) :-
  get_section(Toklist,Sents,Rest,Sect,Printname),!, %get section of report
  parse_sentences(Sents,Fmtlist,Failed,Undef,UnSent,
  Sect,Mode,Exam,_,_),
  process_sects(Rest,mixed,Mode,Exam,Fmtlist2,Fail2,Undef2,
  UnSent2,_).

(outputformat(htext), 1, Fmt0 = [[section [Printname]]], % name of section
Fmt0 = []),
append([Fmt0,Fmtlist,Fmtlist2],Fmtlists), % list of formatted output
append([Failed,Fail2,Fails]), % list of sentences having parsing errors
append([Undef,Undef2,Undef]), % list of words not in lexicon
append([UnSent,UnSent2,UnSents]). % list of sentences with undefined words.

% Input is a single section
process_sects([Toklist,Sect,Mode,Exam,Fmtlist,Failed,Undef,
  UnSent,_]) :-
  Sect \= mixed,
  retractall(paragno(_)),
  assert(paragno(1)),
  retractall(sentno(_)),
  assert(sentno(0)),
  parse_sentences(Toklist,Fmtlist,Failed,Undef,UnSent,
  Sect,Mode,Exam,_,_),
  (outputformat(htext), Fmtlist1 \= [], !,
  append([[section,[Sect]]],Fmtlist1,Fmtlist),
  outputformat(htext), Fmtlist = [], !, Fmtlist = [[section,[Sect]]]];
  Fmtlist = Fmtlist1
).

```

```

outputresults(Fmtlist0,Failed,Errfile,Undef,Unfile,UnSent,Outfile,
  Amount,Type,Exam,Compno,DocComp,NewCompno,Caller,Protocol) !-
  (Amount = full, Fmtlist = Fmtlist0, !, q do nothing to formatted output
  Amount = short, Fmtlist = Fmtlist0, !, q remove modifiers (Fmtlist0,Fmtlist1). !-
  q remove neg and/or old findings
  member(Amount,[pac,pos]), removes frontarg(Amount,Fmtlist0,Fmtlist1). !-
  fixuplists(Fmtlist1,Fmtlist,Type) qproduces extra brackets
  ),
  (Protocol = html, !, OP = html,
  Caller = server, !, OP = html,
  OP = plain),
  (Type = nested, !, write(Fmtlist), new_line(OP),
  (Type = line, !, writelnlines(Fmtlist,OP), new_line(OP), !,
  Type = htext, !, markupsents(Fmtlist), new_line(OP), !,
  Type = indented, !, nl, writeindentform(Fmtlist,Type,OP), !,
  Type = h1l, !, write('<h1>'), !,
  outputhl7(Fmtlist,Exam,Outfile,Compno,DocComp,NewCompno,OP),
  outputhl7(Fmtlist,Exam,Outfile,Compno,DocComp,NewCompno,OP),
  write('</h1>'), !, end of h1 part of output
  write('<line>')
  (Fmtlist = [], writelnlines(Fmtlist,OP), !,
  !, true
  ), q write non-empty
  write('</line>'), nl, !, qnested part
  outputhl7(Fmtlist,Exam,Outfile,Compno,DocComp,NewCompno,OP) qdefault

```

-9-

```

),
(Caller = server,
write_message(Unfile, undef, Caller, '<HR><H2> Undefined Words </H2>')
,
Caller = exe, undef \= [],
write_message(Unfile, undef, Caller, '***** Undefined Words *****')
%write_highlight([], undef, Caller)
,
true
),
%(
 Caller = server, Unsent \= [], undef \= [], !,
 write('<P> Undefined Words are Highlighted in Sentences Not parsed.')
,
true),
%(
 undef = [], !, %no undefined words
% Unfile = [], !,
% tell(Unfile),
% write('**** Undefined Words in domain '),
% write(Exam), write('****'), nl,
% write(Undef), nl
% write_sentences(Unsent)
),
(Caller = server,
write('<HR><H2>Sentences/Phrases Not parsed </H2>'), !,
write_highlight(Undef, Unsent, Caller),
write_highlight([], Failed, Caller)
,
Caller = exe, Errfile \= [], Failed \= [],
tell(Errfile),
write('**** Sentences/Phrases Not parsed *****'), nl,
%write_highlight(Undef, Unsent, Caller),
%write_highlight([], Failed, Caller)
,
true % no Errfile to write to
),

```

```

(OP = html, 1, write('</body></html>'), & end document
true).

  (Failed = [], 1, &no parsing failures
  Errfile = [], 1, & don't write problem messages
  tell(Errfile),
  & write('*** Grammar Failures in domain '),
  & write(Exam), write('****'), nl,
  write_sentences(Failed), nl,
  ).

  set_args: Process options
  Argument options
  -d Amount of output desired. The choices are:
    full (no filtering of output is done)
    pos (only positive findings are shown)
    pac (only current and positive findings are shown)
    short (certain modifiers are not shown)
  -c Component Number (starting component number for HL7 output
    default is 1)
  -d Component where document number is stored (default is 0)
  -e Examtype (otherwise default is chestxray)
  -p Probfile (otherwise default is Problem messages are not written to file)
  -i Infile (if input is supplied by file and not standard input
  -s Section (default is impression)
  -m Mode (default is bp; the 6 choices are bp, model, nodes5
  -o Outfile (if output should be file and not standard output)
  -? Provide list of default arguments
  -f or -t Type of output (default is h17, the choices are:
    h17, nested, line, h171, htext)

```

```

* The line version has nesting but consists of single
* unconnected findings;
* The hl7 version has both hl7 + line; each is tagged
* -pr Protocol - html or Plain
* -u Undefs (otherwise default is - undefined messages are not written
* to a file)
set_args (Args, Examtype, Section, Mode, Infile, Outfile,
          Amount, Type, Component, Startdoc, Prbfile, Undef, Protocol) :-
    set_examtype (Args, Examtype),
    set_section (Args, Section, Examtype),
    set_mode (Args, Mode),
    set_amount (Args, Amount),
    set_protocol (Args, Protocol),
    set_infile (Args, Infile),
    set_outfile (Args, Outfile),
    set_type (Args, Type),
    set_comp (Args, Component),
    set_startcomp (Args, Startdoc),
    set_prbfile (Args, Prbfile),
    set_undefs (Args, Undef).

set_examtype (Args, Examtype) :-
    q .- e or e option specifies exam type
    (nextto('`-e`', B, Args), nextto('`e`', B, Args)), !,
    examis (B, Examtype).

examis (cheatxray, cheatxray) :-
    examis (B, cheatxray) !.

examis (B, cheatxray) :-
    & MEDCODES for descendants of cheatxray3
    member (B, [`10220`, `10221`, `27489`, `27619`, `27620`,
               `10220`, `10221`, `27489`, `27619`, `27620`,
               `27621`, `27622`, `27624`, `27625`, `27626`, `27627`,
               `27621`, `27622`, `27624`, `27625`, `27626`, `27627`,
               `27628`, `27629`, `27628`, `27629`,
               `27630`, `27631`, `27632`,
               `27630`, `27631`, `27632`,
               `40607`, `40676`, `40680`, `40682`, `40684`, `40689`,
               `40607`, `40676`, `40680`, `40682`, `40684`, `40689`,
               `40692`, `40744`, `40759`, `40762`, `40764`, `40829`,
               `40692`, `40744`, `40759`, `40762`, `40764`, `40829`]),
    !.

```

```

'40862', '40863', '40877', '40880',
40862, 40863, 40877, 40880,
'40910', '40923', '40924', '40933', '40934', '40951', '40952',
40910, 40923, 40924, 40933, 40934, 40951,
40952,
'40966', '40969', '40974', '41012', '41016', '41017', '41852',
40966, 40969, 40974, 41012, 41016, 41017,
41852,
'41854', 41854], 1.

examis (mammography, mammography) :- 1.
examis (mammogram, mammography) :- 1,
examis (mammograms, mammography) :- 1.

examis (E, mammography) :-  

  & MBDCODES for descendants of mammograms
  member(IB, ['40728', '40778', '40864', '40966', '42264', '27623', '27625',
  40728, 40778, 40864, 40966, 42264, 27623, 27625,
  '40863', '33920', 40863, 33920]), 1.

examis (daum, daum) :- 1.
examis (oprpt, oprpt) :- 1.
examis (echo, echo) :- 1.
examis (__, []).

set_comp(Arg, Component) :-  

  & -c or c option
  (nextto(' -c', C, Args); nextto(c, C, Args)),
  name(C, Lchars), number_chars(Component, Lchars). %convert to number
  set_comp(__, 1). %default component number for h17

```

-10-

```

set_startcomp(Args, Startdoc) :-
  (nextto('`-d', D, Args), nextto(d, D, Args)), !, & -s or s option
  name(D, Dchars), number_chars(Startdoc, Dchars).

set_startcomp(_, 0). & default parent finding

set_section(Args, Section, Examtype) :-
  (nextto('`-s', S, Args), nextto(s, S, Args)), !, & -s or s option
  sectionis(S, Section, Examtype).

set_section(., 'report impression item', chestxray). & default section
set_section(., 'report impression item', mammography). & default section
set_section(., 'report clinical information item', dsum). & default section
sectionis('34900', 'report description item', __) :- !.
sectionis('34900', 'report description item', __) :- !.
sectionis('34901', 'report description item', __) :- !.
sectionis('34901', 'report impression item', __) :- !.
sectionis('34901', 'report impression item', __) :- !.
sectionis(impression, 'report impression item', __) :- !.
sectionis('34902', 'report clinical information item', __) :- !.
sectionis('34902', 'report clinical information item', __) :- !.
sectionis('clinical information', 'report clinical information item', __) :- !.
sectionis('clininfo', 'report clinical information item', __) :- !.
sectionis('clinical', 'report clinical information item', __) :- !.
sectionis(mixed, __) :- !. & section headers are included in text
sectionis(testing, testing, __) :- !.
sectionis('report impression item', mammography) :- !. & default section
sectionis('report impression item', chestxray) :- !.
sectionis('discharge diagnoses', 'report discharge diagnosis item', __) :- !.
sectionis('history of present illness', 'report history of present illness item', __) :- !.
sectionis('chief complaint', 'report chief complaint item', __) :- !.
sectionis('past medical history', 'report past medical history item', __) :- !.
sectionis('review of systems', 'report review of systems item', __) :- !.
sectionis('hospital course', 'report hospital course item', __) :- !.
sectionis('physical examination', 'report physical examination item', __) :- !.
sectionis('social history', 'report social history item', __) :- !.
sectionis(X, dsum) :- !.

```

```

set_mode(Argv,Mode) !:
  (nextto('`',M,Argv), nextto(m,M,Argv)) . 1,
  mode is (M,Mode) . 1.
  set_node( _, bp). & default output type

mode is (relax,mode2) . 1.
mode is (strict,mode1) . 1.
mode is (skip,mode4) . 1.
mode is (longest,mode3) . 1.
mode is (best,bp) . 1.
mode is (mode1,mode1) . 1.
mode is (mode2,mode2) . 1.
mode is (mode3,mode3) . 1.
mode is (mode4,mode4) . 1.
mode is (mode5,mode5) . 1.

set_type(Argv,Type) !:
  (nextto('`-t',Type,Argv), nextto('`t',Type,Argv)) ;
  nextto('`-f',Type,Argv), nextto('`f',Type,Argv)
  ) . 1, & -f -t, or t option
  member(Type, [nested, line, indented, h17, h171, drg, summary, htext]), 1.
  set_type( _, h17). & default form
  set_amount(Argv,Amount) !:
  (nextto('`-a',Amount,Argv), nextto('`a',Amount,Argv)) . 1, & -a or a option
  member(Amount, [full, short, pac, pos]), 1.
  set_amount( _,full). & default type
  set_protocol(Argv,Protocol) . 1.

```

```

(nextto('pr',Protocol,Args), nextto('pr',Protocol,Args)),
member(Protocol,[html,Plain]), !.

set_protocol(_,Plain),
set_undefs(Args, Undefs) !-
  nextto('u', Undefs,Args), nextto(u,Undefs,Args), !. & undef file option
set_undefs(_,[]). & default is no file of undefineds created

set_infile(Args,Infile) !-
  nonvar(Infile), !. & Infile is set already
  nextto('i', Infile,Args), !.
  nextto(i,Infile,Args), !.
  set_infile(_,user_input). & default is standard input

set_prbfile(Args,Prbfile) !-
  nextto('p',Prbfile,Args), !. nextto(p,Prbfile,Args), !. & prob file option
set_prbfile(_,[]). & default is no file of problems is created

set_outfile(Args,Outfile) !-
  nonvar(Outfile), !. & Outfile is already set
  nextto('o',Outfile,Args), !. nextto(o,Outfile,Args), !. & outfile option
set_outfile(_,user_output). & default is standard output

new_line(html) :- write(<br>), !.
new_line(server) :- write(<br>), !.
new_line(exe) :- nl.
new_line(plain) :- nl.

write_message(_,[],exe) :- !.
write_message([_|_],exe) :- !.
write_message(_,[],plain) :- !.
write_message([_|_],plain) :- !.
write_message(File,Contents,Caller,Message) :-
  ( member(Caller,[exe,plain]), tell(File), !,
  true),
  write(Message), new_line(Caller),
  (Contents = [], write(Contents), new_line(Caller))

```

```

write_highlight([_, _, _]) :- !.
write_highlight(Undef, [Word|UnSent], Caller) :- !,
  member(Caller, [html, server]),
  (member(Word, Undef), write('<strong>'), write(Word), write('</strong>'))
  ; write(Word),
  (sentend([Word], Caller), write_highlight(Undef, UnSent, Caller), !;
  write(' '), write_highlight(Undef, UnSent, Caller)).
write_highlight(Undef, [Word|UnSent], Caller) :- !,
  member(Caller, [exe, plain]),
  is_list(Word), !, % bracketed phrase
  write_highlight(Undef, Word, Caller), % write out words in phrase
  write(' '), write_highlight(Undef, UnSent, Caller).
write_highlight(Undef, [Word|UnSent], Caller) :- !,
  member(Caller, [exe, plain]),
  write_highlight(Undef, UnSent, Caller).
write_highlight(Undef, [Word|UnSent], Caller) :- !,
  member(Caller, [exe, plain]),
  write(Word), % write individual word
  (sentend([Word], Caller), write_highlight(Undef, UnSent, Caller), !;
  write(' '), write_highlight(Undef, UnSent, Caller)).
write_highlight([_, _, plain]) :- !.
write_highlight([_, _, exe]) :- !.
sentend([X|_], Caller) :- !,
  member(X, ['.', ',', '?']), new_line(Caller), !.
removemodifiers([ ], [ ]) :- !.
removemodifiers([ [sentence, [ ]], [ ] ], [ ]) :- !, % at end of file marker
removemodifiers([ [section, S1|Fmts], [ [section, S1|NewFmtS] ] ], [ ]).
removemodifiers([ [FmtS, NewFmtS] ], [ ]).
removemodifiers([ S1|Rest], [ NewS1|NewRest] ) :- !,
  removemods(S1, NewS1).

```

-11-

```

removemodifiers(Rest, NewRest), !.
removemods(+Pmts, -NewPmts): removes certain modifiers from
  Pmts when mode is "short" form, resulting in NewPmts
removemods([], []): !.
removemods([[], []]): !.
removemods([[], [sentence, s] | Pmts], [[sentence, s] | NewPmts]): !.
removemods([[], [sentence, s] | Pmts], [sentence, s] | NewPmts): !.
removemods(Pmts, NewPmts), !.
removemods([Pmt1 | Rest], [NewPmt1 | NewRest]): !.
chkandremove(Pmt1, NewPmt1),
removemods(Rest, NewRest), !.
removemods([finding, demo[L] | Rest], [finding, demo[L] | NewRest]), !.
removemods([finding, demo[L] | Rest], [NewPmt1 | NewRest]), !.
removemods([Pmt1 | Rest], [NewPmt1 | NewRest]), !.
chkandremove([Pmt1 | Rest],
  &removefromlist(Pmt1, NewPmt1),
  &removemode(Rest, NewRest), !. % remaining ones
removemode(Rest, NewRest), !.
removemode([[], []]), !.
removemode([Pmt1 | Rest], [NewPmt1 | NewRest]), !.
chkandremove(Pmt1, NewPmt1), !. % formats
removefromlist(Rest, NewRest), !.
chkandremove([Type | Rest], [Type | Rest]), !. % leave alone
member(Type, [sentence, section]), !.
chkandremove([finding, demo | Rest], [finding, demo | Rest]), !. !. % leave alone
chkandremove([Type | _, []]), !. % remove certain type of formats first
skipTypes(Type), !.
chkandremove([Type, Value], [Type, Value]), !. !. % no mods to remove
chkandremove(Fmt, NewFmt), !.
removemode(Fmt, NewFmt),
removemode(+OneFmt, -NewFmt): !.
removemode([Type, Value | Mods], [Type, Value | NewMods]): !.
processmods(Type, Mods, NewMods), !.
processmods([[], []]), !.
removemode([[], []]), !.
processmods([_, [], []]), !.
processmods(Type, [Mod1 | Rest], NewMods): !.
Type \= finding,
  chmod1(Mod1, NewMod),

```

```

processmods([Type, Rest, NewMod]) :-  

  ( NewMod \= [], append([NewMod], NewMod1, NewMod) ;  

    NewMod = [] , NewMod1 = NewMod ) .  

processmods([finding, [Mod1|Rest], NewMod]) :-  

  chkmod2([Mod1, NewMod]) .  

processmods([Type, Rest, NewMod1]) :-  

  ( NewMod \= [], append([NewMod], NewMod1, NewMod) ;  

    NewMod = [] , NewMod1 = NewMod ) .  

chkmod1([Type|Rest], NewMod) :-  

  okmod1([Type], NewMod = [Type|Rest] ),  

  NewMod = [] .  

chkmod1([], []) .  

chkmod2([Type|Rest], NewMod) :-  

  okmod2([Type|Rest] ),  

  okmod2([Type], NewMod = [Type|Rest] ),  

  NewMod = [] .  

chkmod2([], []) .  

skiptypes([Type]) :-  

  member([Type, [status, prev_info, reltime, normalfinding, demo] ],  

        okmod1([Type]) ) .  

member([Type, [status, certainty, bodyloc, descriptor, region, sentid,  

              member([Type, [status, certainty, bodyloc, descriptor, region, sentid,  

                'section of examination'] ] ) .  

okmod2([Type]) :-  

  member([Type, [status, certainty, bodyloc, descriptor, region, sentid,  

    degree, change,  

    'section of examination', age, ethnic, sex, race] ],  

        removefromarg(_, [sentence, ['11'].[]] ) ) .  

removefromarg(_, [sentence, ['11'].[]] ) :- !.

```

```

removefromtarg(Amount, [S1|Rest], Rem) :-  

  removefromtarg(Amount, S1, NewS1),  

  removefromtarg(Amount, Rest, NewRest),  

  (NewS1 \r [ ], Rem = [NewS1|NewRest], !;  

  Rem = NewRest, !).  

removefromsent( _, [section,S], [section,S] ) :-  

  retractall(currentsect( _ )),  

  assert(currentsect( S )).  

assert(currentsect( S )) :-  

  removefromsent( _, [section,S] ), !.  

removefromsent( _, [currentsect( _ )],  

  retractall(currentsect( _ )),  

  assert(currentsect( S ))).  

removefromsent(Amount, [lsentence,B] | Pmts], [lsentence,S] | NewPmts]) :-  

  removefromings(Amount, Pmts, NewPmts), !.  

removefromsent( _, [ ], [ ] ) :- !.                                % add extra bracket when there is  

removefromsent(Amount, Pmts, NewPmts) :-  

  removefromings(Amount, [Pmts], NewPmts), !.                      % no sentence prefix  

removefromings(Amount, Pmts, Pmts).  

removefromings( _, [ ], [ ] ).  

removefromings(Amount, [P1|Rest], Rem) :-  

  removefromings(Amount, P1, NewP1),  

  removefroming(Amount, P1, NewP1),  

  removefromings(Amount, Rest, NewRest),  

  (NewP1 \r [ ], Rem = [NewP1|NewRest], !;  

  Rem = NewRest, !).  

removefroming(Amount, [P1|Rest], [F1|Rest]) :-  

  chktypesandpacs(Amount, [P1|Rest]), !.                          % these formats are positive, curr, not  

removefroming( _, F, [ ] ).                                         % this format is normal, negative, or old  

%chktypeandpacs([Type|_]) :-  

%  member(Type, [sentence,section]), !.                            % leave these alone  

%chktypeandpacs(Amount, [Type,_,|Mds]) :-  

%  chktargtype(Amount, [Type, Mds]),  

%  chkpacs(Amount, Mds).  

%chkargtype(Finding, Mds) :-  

%  \+ chkfnormal(Mds), !.                                         % change mod = better, descr mod = normal  

%chkargtype(Type,_) :-  

%  \+ member(Type, [normalfinding, status, date, timeper, prev_info]).  


```

```

chkpacs (__, []) :- !.
chkpacs (Amount, Mods) :- !, chkforpos (Mods), !, % remove neg. only, don't filter time
  ( Amount = pos, !;
  chkforsect (Mods),
  chkforcurdate (Mods) ) .
chkfornormal (Mods) :- !.
(setof (X, member ([change, X], Mods), Changes),
 (outputform (htext), gettargets (Changes, Newchanges), !; % target form
  Newchanges = Changes),
 member ([better], [Newchanges]), !,
 setof (X, member ([descriptor, X], Mods), Descrs),
 (outputform (htext), gettargets (Descrs, Newdescrs), !;
  Newdescrs = Descrs),
 member (normal, Newdescrs), !).
chkforpos (Fmt) :- !.
(setof (X, member ([certainty, X], Fmt), Certs), !, % certainty modifiers
 (outputform (htext), gettargets (Certs, Newcerts), !;
  Newcerts = Certs),
 \+ isneg (Newcerts), !;
 true).
chkforcurstat (Mods) :- !.
(setof (X, member ([status, X], Mods), Status), !, % status modifiers
 (outputform (htext), gettargets (Status, Newstatus), !;
  Newstatus = Status),
 \+ isold (Newstatus), !;
 true).
chkforsect (Mods) :- !.
\+ outputform (htext), !. % leave out if past history section

```

```

\+ ispasthistory(Mods) . !,
chkforcurstat(Mods) . !, % not past history - check status for current
chkforact(Mods) . !, % htext mode, check status for current
outputform(htext),
( ispasthistory(Mods) , !, % leave alone if in past history section
  chkforcurstat(Mods)
).

ispasthistory(Mods) :-
  member(['section of examination',X], Mods),
  X = 'past history report item', !.

ispasthistory(Mods) :-
  currentsect(S),
  member(S, !, past history report item, ('PAST MEDICAL HISTORY', !)).

chkforcurdate(Mods) :-
  \+ chkforcurdate(Mods),
  \+ chkforpastadm(Mods).

chkfordate(Mods) :-
  member([date|_], Mods), % dates are generally in the past
  chkforpastadm(_) . !, fail.
  gettargets([], [], !).
  gettargets([W1|Rest], [T1|Trest]) :-
  foundword(W1, _, T1),
  gettargets(Rest, Trest), !.

isneg(X) :-
  intersect(X, [no, negative, deny, 'rule out']). !.

isold(X) :-
  intersect(X, ['risk factor', resolved, end, previous, 'exposed to',
    'positive family history', future, removed, need, prepare, post,
    'negative family history', hold, 'past history', postoperative,
    prepare, 'ran out', 'treatment option', 'strong family history',
    tendency, weaned, withdrawal, avoid, inactive, need, offer, plan, soon]). !.

% convert output to database form and write out h17 with codes

```

```

outputh17(Fmtlist, Exam, Outfile, Compno, DocComp, NewCompono, Caller) :-  

  xformtodb (Fmtlist, NewFmtlist, Exam), !, % database form  

  writeh17(NewFmtlist, Outfile, Compno, DocComp, NewCompono, Caller). % h17

fixuplists([],[],_). :- !.  

fixuplists(P,F,htext) :- !, !. % output is correct for markup  

fixuplists([F1|Rest],Fixed,Type) :- !, % remove extra bracket around sentences
  fixuplists(Rest,NewRest,Type),  

  append([F1,NewRest],Fixed), !.  

% write out one finding in nested form per line
writelns([],_) :- !, !. % no more findings
writelns([Onefinding|Morefindings],Caller) :-  

  write(Onefinding), new_line(Caller),  

  writelns(Morefindings,Caller), !.  

writelns([Morefindings],Caller), !.  

writelnform2([],_) :- !.  

writelnform2([_|_],_) :- !.  

writelnform2(Findings,Type,html) :-  

  write('<pre>'), nl,  

  writeindentfindgs(Findings,Type,plain),  

  write('</pre>'), nl.  

writelnform2(Findings,Type,plain) :-  

  writeindentfindgs(Findings,Type,plain), !.  

writelnfindgs([],_) :- !.  

writelnfindgs([F1|Rest],Type,Op) :-  

  writeIndented(F1,Type,Op),  

  writelnfindgs(Rest,Type,Op), !.  

writelnfindgs(Finding,Type,Op) :-  

  write('<pre>'), nl,  

  writeIndented(Finding,Type,Plain), !, % in pre mode - new line is recognized  

  write('</pre>'), nl.  

writelnform(Finding,Type,Plain) :-  

  writeIndented(Finding,Type,Plain), !.  

writelnIndented(Finding,Type,Plain). % in html

```

```

writeIndented([,---]) :- !, !. % no more findings
writeIndented([Onefinding|Morefindings],Type,Caller) :-
  writeInd(Onefinding,Type,Caller), writeIndented(Morefindings,Type,Caller), !.
writeInd([,---]) :- !.
writeInd([failure],_,Caller) :- write('...parsing failure...'), !,
  nl, !.
new_line(Caller), !.
writeInd([Type,Finding|Mods],Output,Caller) :-
  (Output = indented; Output = summary), !,
  write(Type), write(':'), write(Finding),
  nl,
  new_line(Caller),
  writeMods(Mods, nl, Caller),
  nl.

%new_line(Caller).

% while testing DRGs write out findings relevant to DRGs only
writeInd([finding,Finding|Mods],drg,Caller) :-
  drgchk(Finding), !, % Finding relevant for DRG
  checkMods(Mods), !, % check that Mod is not certainty = 'no'
  writeInd([finding,Finding|Mods], indented, Caller).
writeInd(_,drg,_) :- !, !, % no output for this finding
writeMods([,---],_,_) :- !, !, % no more mods
writeMods([OneMod|MoreMods],Tabno,Caller) :-
  writeOneMod(OneMod,Tabno,Caller), writeMods(MoreMods,Tabno,Caller), !.
writeOneMod([,---],_) :- !.
writeOneMod([Mod,Value|Rest],Tabno,Caller) :-
  writeBlanks(Tabno), write(Mod), write(' > '),
  write(Value),
  nl, !.
new_line(Caller), !,
(Rest = [] ; Newtab is Tabno + 8, writeMods(Rest,Newtab,Caller)).

writeOneMod([L,Tabno,Caller]) :-
  strip(L,Newl), !, writeOneMod(Newl,Tabno,Caller).
writeOneMod([,---],_) :- !.
writeBlanks(0) :- !.
writeBlanks(Tabno) :- !.

```


-13-

```

write_markedsent ([W1,W2,W3|S]) :-  

  W1 = '<b><font color =',  

  write(W1), write(W2), write(' '), write(W3), write(' '),  

  write_sentences(S).  

write_markedsent(S) :- write_sentences(S).  

markupformats ([S, [], S) :- !, !. % no markup is needed  

markupformats ([S, [Fmt1|Rest], MSent) :-  

  color_markup (S, Fmt1, Rest), MSent).  

color_markup (S, Fmt1, MS1),  

  markupformats (MS1, Rest, MSent), !. % markup 1st part  

  markuprest  

%color_markup (4L, +Fmt, +Fmt):  

color_markup (L, [], L).  

color_markup (L, [finding, Value|Mods], NewL) :-  

  Type = finding, Value \= demo,  

  Type = finding, Value = demo,  

  Color = "brown",  

  (setof (X, member ([change, X], Mods), Changes),  

  markup_list ([L, Changes, Color, NewL]), !,  

  setof (X, member ([degree, X], Mods), Degrees),  

  markup_list ([L, Degrees, Color, NewL]), !,  

  L = NewL),  

  (setof (X, member ([descriptor, X], Mods), Descrs),  

  markup_list ([NewL1, Descrs, Color, NewL2]),  

  NewL2 = NewL1),  

  write_color (NewL2, Value, Color, NewL).  

% Finding  

color_markup (L, [Type, Value|Mods], NewL) :-  

  (Type = problem, Color = "brown", NL = L),  

  Type = med, medcolor (L, Mods, NL, Color), !,  

  !. % differentiate meds from  

  !. % reaction to meds  

Type = procedure, Color = "blue", NL = L,  

Type = device, Color = "blue", NL = L,  

(Type = procedure, gettargets ([Value], Proc),  

  member (Proc, [view, examination]), !, !, % dont markup generic procedure  

  write_color (NL, Value, Color, NewL)).  

%color_markup (L, [finding, demo|Mods], NewL) :- %get age, sex, race, ethnic

```

```

color_markup(L,_,L).
medcolor(L,Mods,NL, "brown") :-
    currentsect(B), member(S, [ 'report allergy item', 'ALLERGIKS:.' ]), !,
    (Mods = [], L = NL, !),
    chkforreact(L,Mods,NL),
    medcolor(L,_,L, "green"), !, !. %not allergy section - medication
medcolor(L,Mods,NL, "brown") :-
    chkforreact(L,Mods,NL).

chkforreact(L,Mods,NL) :- % markup up reaction to med also
    setof(X, member([reaction,X],Mods), React),
    gettargets(React,Tarreact),
    listhasreact(Tarreact), markreact(React,Tarreact,L,NL),
    L = NL.

medcolor(L,_,L, "green"). % default
markreact([],[],L,L) :- !.
markreact([R|Rest], [Tar|TarRest], L, NL) :-
    (isreact(Tar), write_color(L,R, "brown"), NewL),
    markreact(Rest,TarRest,NewL,NL), !,
    markreact(Rest,TarRest,L,NL), !.
markreact(_,_,L,L).

isreact(Tar) :- % adverse reaction
    member(Tar, [allergy,intolerant,'not effective','toxic','side effect'],
          'not better',reaction)).
listhasreact(Tarreact) :- % not better, reaction).
    intersect(Tarreact,[allergy,intolerant,'not effective',toxic,'side effect'],
              'not better',reaction)]).

% markup_list(+sentlist,+Wordlist,+Color,-NewL)
markup_list([L,_|Ls], NewL) :- !, !. % no more elements to markup
markup_list([L, [Value|Rest], Color, NewL]) :-

```

```

write_color(L,Value,Color,NewL),
markup_list(NewL,Rest,Color,NewL) :-.
write_color([L,Value,Color,NewL]) :-.
atomic([Value]), !, suffix([L,[Value|Rest]]), !, % word in sentence list
append([Value|Rest],L), !, % part of sentence before word
append([Prefix],[Value|Rest],L), !, % check word is not marked already
(notmarked(Prefix), !, % check word is not marked already
surround([Value,Color,MVal]), append([Prefix,MVal,Rest],NewL));
write_color([Rest,Value,Color,NewR]), !, % mark up latter part of sentence
append([Prefix],[Value|NewR],NewL),
append([Prefix],[Value|NewR],NewL), !, % add to first part
L = NewL % don't do anything
).

write_color([L,Value,Color,NewL]) :-.
Value = [W1|_], suffix([L,[W1|Rest]]), !, % word of phrase in sentence
append([Prefix,[W1|Rest],L], !, % prefix is part of L before phrase
(notmarked(Prefix), !, % phrase is not marked already
last([Wn,Value]), suffix([Rest,[Wn|Restn]]), !, % last word in phrase
add_colorprefix([W1,Color,MW1], !, % markup before W1 of phrase
append([Phrase,[Wn|Restn],Rest], !, % get rest of phrase in sentence
add_colorsuffix([Wn,MWn], !, % markup up and of phrase
append([Prefix,MW1,Phrase,MWn,Restn],NewL), !, % put all pieces together
write_color([Rest,Value,Color,NewR]), !, % write up latter part of sentence
append([Prefix,[W1|NewR],NewL], !, % leave sentence unmarked
append([Prefix,[W1|NewR],NewL], !, % leave sentence unmarked
write_color([L,_,_,L]),
surround(W,Color,CList),
add_colorprefix(W,Color,PreL),
add_colorsuffix(PreL,CList),
add_colorprefix(+W,+Color,-CW), !, % CW is word, CW is a list containing
% starting markup for color (i.e. markup preceded W)
add_colorprefix(W,Color,['<b><font color = '|CLint|] ),
append([([color],[`size=+1`],[W]),CList],
add_colorprefix(+L,-W), !, L is a list, W is a list containing ending
% markup for a color which is in a list preceded L in sentence
add_colorsuffix(L,SL),
append(L,['</font></b>'],SL).

```


-14.-

```

    Newform = []
    ; % is not a finding - don't produce output
    Newform = []
),
xformtodb(Rest, Newrest, Examtype),
append(Newform, Newrest, Dbform).

isonefinding(X) :-
    X = [Y|_],
    atom(Y).

ismultifinding(X) :-
    X = [Y|_],
    list(Y).

tdbformult([],[],[]):-!.
tdbformult([X|Rest],Xdblist,Examtype) :-
    tdbform(X,Xdb,Examtype), !, % single finding or recommend
    tdbformult(Rest,Xdbrest,Examtype),
    Xdb = [], Xdblist = Xdbrest,
    append([Xdb],Xdbrest,Xdblist).

tdbform([Type,Finding|Mods],Xdb,Examtype) :-
    okframetype(Type), Type \= recommend, !,
    ( computeicode(Finding,finding,Medterm,Examtype), %find medcode
    certaintychx(Mods,Cval),
    statuschk(Mods,Cval),
    %removenull(Mods,Mod$1),
    fixmods(Mods,Mod$1,Examtype),
    computeval(Cval,Sval,Value),
    Xdb = [Medterm,Value|Mod$1],
    Xdb = [] ) %finding cannot go in database - no medcode
    ). !.

```

```

todbform([recommend,PubExam,Mod1,Xdb,Examtype) :-  

  computerecval(PubExam,Mod1,Val,Examtype,Mod2), % compute value for recommend  

  fixmods(Mod1,Mod2,Examtype),  

  &removemod(Mod1,Mod2),  

  Xdb = ['report recommendation item',Val|Mod2],  

  Xdb = [] %finding cannot go in database - no modcode  

), !.  

todbform(_,[],_) :- !.  

  % compute value of finding for database  

  findingval(Medterm,Cval,Newmods,Xdb) :-  

  computerecval(Cval,Cval,Value),  

  Xdb = [Medterm,Value|Newmods].  

  % Value of recommendation is computed as follows:  

  1. If there is an exam or procedure mentioned, that value  

  2. Otherwise, if 'follow up' is value and/or no exam is  

  is mentioned, the value will be the current exam.  

  3. Otherwise, if there is a follow up time period mentioned  

  the value will be the current exam, and it will have a  

  modifier which is follow-up times, with a value which  

  is the follow up time.  

computerecval(PubExam,Mod1,Val,Examtype,Mod2) :-  

  certainychk(Mod1,Cval),  

  degreechk(Mod1,Dval),  

  getspecific(PubExam,SpExam), % get most specific exam  

  % only followup is mentioned, recommend denotes followup

```

```

(spExam = 'followup', computocode(Examtype, Val, Examtype); %code for exam
\+ spExam = 'followup', %followup procedure is stated
  % value is a more specific procedure
  (computocode(spExam, Type, Val, Examtype), Val = Examtype)
).

% if there are modifiers - check time period and add approp. modifier
addtimemod(Mods, NewMods), !.

% Get specific examination - not word followup
getspecific(PuExam, FuExam),
atom(PuExam), !.

% recommend a list of exams - get most specific
getspecific(PuExam, SpExam),
is_list(PuExam),
PuExam = [Head|Tail],
( Head = followup, !, getspecific(Tail, SpExam),
  SpExam = Head
).

addtimemod(Mods, NewMods),
futexamchk(Mods, Futval), % get value for future exam if it exists
Futval = [unitval,X,Timeunit], %calculate standard time
% This will be replace when number+unit can be stored in db
(X = 6, Timeunit = month, FuModval = 'follow-up in 6 months', %6 month foll
owup
X = 6, Timeunit = week, FuModval = 'follow-up in 6 weeks', %6 week foll
owup
X = 1, Timeunit = year, FuModval = 'follow-up in 1 year'),
append(Mods, [!,'follow-up times', FuModval]), NewMods),
!.

addtimemod(Mods, Mods).

computocode(Regterm, Type, Medterm, Examtype),
(synonyms(Regterm, T, Medterm, Domain), % There is a Medterm for this term
% general domain matches any exam; otherwise Examtype must match
( Domain = general, Domain = Examtype, number(Examtype, Domain) ),

```

```

( T = Type, !; & same type of information
  subtype(Type,T), !; & Type is a subtype of T
  member(Type, [procedure,problem,device,normalfinding,postprocedure,
    allergy]), T = finding &equivalent
  )

  Medterm = Regterm & if there is no synonym, use original
  medcode(Medterm,_). & can go into db form only if there is a medcode

certaintychk([],[]):-.
certaintychk([X|Rest],Val) :-.
X = [certainty,Val|_],
certaintychk(Rest,Val).

statuschk([],[]):-.
statuschk([X|Rest],Val) :-.
X = [status,Val|_],
statuschk(Rest,Val).

degreechk([],[]):-.
degreechk([X|Rest],Val) :-.
X = [degree,Val|_],
degreechk(Rest,Val).

futurechk([],[]):-.
futurechk([X|Rest],Val) :-.
X = [future_exam,Val|_],

```

& check if modifier contains a recommended time for future exam

~~futurechk([],[]):-.
futurechk([X|Rest],Val) :-.
X = [future_exam,Val|_],~~

-15.-

```

futexamchk(Rest,Val).

% remove empty list from present list
fixmod([ ],[ ],_) :- !.
fixmod([( ]|Rest),Newrest,Examtype) :- !,
  fixmod(Rest,Newrest,Examtype), !.
fixmod([X|Rest],Newmod,Examtype) :- !,
  fixmod([X|Rest],Newmod,Examtype), !,
  X = [Type,Regterm|Moremods], !, % type of slot, regularized value, nested mod
  (computecode(Type,slot,Medtype,general), !, % compute slot term
  computecode(Regterm,Type,Medterm,Examtype), !, % compute med term
  fixmods(Moremods,NewMore,Examtype), !, % fix nested mod if any
  NewX = [Medtype,Medterm|NewMore], !, % new controlled vocab term
  NewX = [ ] !, % no controlled term
  ),
  fixmod([Rest,Newrest,Examtype], !,
  (NewX = [ ], !, Newmods = Newrest,
  Newmods = [NewX|Newrest]
  )).

```

```

%computeval([ ],[],'moderate certainty').
computeval(Cval,Sval,Vcode) :- !,
  (\+ Cval = [], Vcode = Cval, !,
  Cval = [ ], \+ Sval = [], Vcode = Sval, !,
  Vcode = 'high certainty').

```

```

writeln7(Findings,_,Compsart,Docstart,Compend,Caller),
findingtoObx(Findings,Compsart,Docstart,Compend,Docstart,Caller).

```

```

findingtoObx([ ],Compsart,Compsart,_,_) :- !.
findingtoObx([Findings|Rest],Compsart,Compend,Docstart,Caller) :- !,
  writeObx(Findings,Compsart,Docstart,Compend,Docstart,Caller),
  findingtoObx(Rest,Compend,Docstart,Caller).

```

```
writeObx([ ], Component, __, Component, __) :- !.
writeObx([Finding, Value| Mods], Component, Parent, Nextcomp, Caller) :-  
    medcode(Finding, Code), medcode(Value, Vcode), !,  
    writeObxPart(Component), !, $OBX|Component|CE|  
    writecoded(Code, Finding),  
    write(Parent), write('|'),  
    writecoded(Vcode, Value),  
    writeendObx(Caller),  
    Newcomponent is Component + 1,  
    !, write modifier component,  
    writeObx(Mods, Newcomponent, Component, Nextcomp, Caller).  
  
% handle unit h17
% writeObx([Finding, Value| Mods], Component, Parent, Nextcomp) :-  
% handle date h17
% writeObx([Finding, Value| Mods], Component, Parent, Nextcomp) :-  
  
% no new component written because Medcode missing
writeObx([_|_], Component, __, Component, __) :- !.  
  
writeObx([ ], Component, __, Component, __) :- !.
writeObx([ [Mod| Rest], Component, Parent, Finalcomp, Caller) :-  
    writeObx([Mod| Component], Parent, Nextcomp, Caller),  
    writeObx(Rest, Nextcomp, Parent, Finalcomp, Caller).  
  
writeObxPart(Component) :-  
    write('OBX|'), write(Component), write('|CE|').  
  
writecoded(Code, Finding) :-  
    write(Code), write('.'), write(Finding), write('|').
```

```
writeandObx :- write('|||||'), nl.  
writeandObx(Caller) :- write('|||||'), new_line(Caller).
```

Claims

1. A computer method for processing medical/clinical data comprising a
5 natural-language phrase,

the method comprising parsing the natural-language phrase and
regularizing the parsed phrase,

wherein parsing comprises referring to a domain parameter whose
value is indicative of a medical/clinical domain from which the data originated.

10

2. The method according to claim 1, further comprising preprocessing the
data prior to parsing, with preprocessing comprising referring to the domain
parameter.

15

3. The method according to claim 1, further comprising encoding at least
one term of the regularized phrase, with encoding comprising referring to the domain
parameter.

20

4. The method according to claim 1, further comprising filtering the
regularized phrase.

5. The method according to claim 1, further comprising referring to an
additional parameter which is indicative of the degree to which subphrase parsing is to
be carried out.

25

6. The method according to claim 1, further comprising referring to an
additional parameter which is indicative of desired filtering.

7. The method according to claim 1, further comprising referring to an
30 additional parameter which is indicative of a desired type of output.

- 17 -

8. The method according to claim 1, further comprising referring to an additional parameter which is indicative of a desired output format.

9. A computer system for processing medical/clinical data comprising a
5 natural-language phrase,

the system comprising means for parsing the natural-language phrase and means for regularizing the parsed phrase,

wherein the parsing means comprises means for referring to a domain parameter whose value is indicative of a medical/clinical domain from which the data

10 originated.

10. The system according to claim 9, further comprising means for preprocessing the data prior to parsing, with the preprocessing means comprising means for referring to the domain parameter.

15

11. The system according to claim 9, further comprising means for encoding at least one term of the regularized phrase, with the encoding means comprising means for referring to the domain parameter.

20

12. The system according to claim 9, further comprising means for filtering the regularized phrase.

13. The system according to claim 9, further comprising means for referring to an additional parameter which is indicative of the degree to which
25 subphrase parsing is to be carried out.

14. The system according to claim 9, further comprising means for referring to an additional parameter which is indicative of desired filtering.

30

15. The system according to claim 9, further comprising means for referring to an additional parameter which is indicative of a desired type of output.

- 18 -

16. The system according to claim 9, further comprising means for referring to an additional parameter which is indicative of a desired output format.

17. A combination of the system according to claim 9 with an interface 5 module for enabling the system to receive input from and/or to produce standardized output for the World-Wide Web and/or a local network.

18. The combination according to claim 17, further comprising means for viewing the output using a standardized browser.

10

19. The combination according to claim 18, wherein the browser is a Web-browser.

1/1

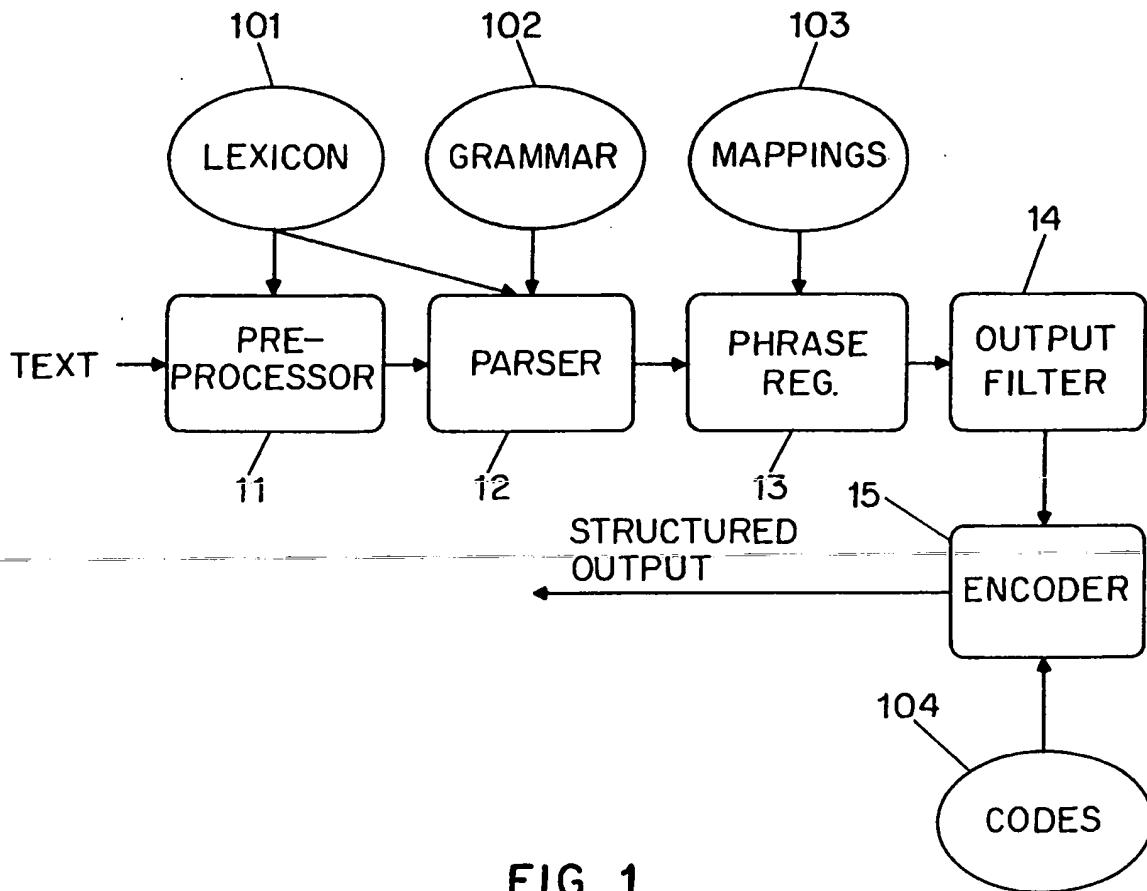


FIG. 1

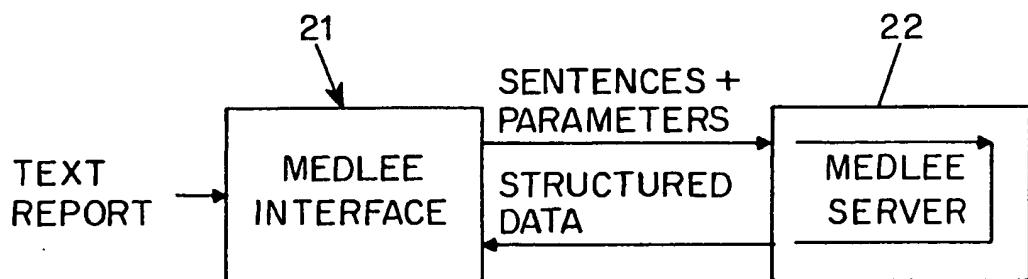


FIG. 2

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US97/19362

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 17/22

US CL :704/9

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 704/1, 9; 705/2, 3

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,377,103 A (LAMBERTI ET AL) 27 December 1994, abstract, figures 1-4; col. 3, line 26 to col. 8, line 18	1-16
A	US 5,265,065 A (TURTLE) 23 November 1993, abstract; figs.4 & 7-10; col. 1, line 9 to col. 3, line 30; col. 8, line 39 to col. 10, line 56; col. 14, line 43 to col. 18, line 33	1-16
A	US 5,251,131 A (MASAND ET AL) 05 October 1993, abstract; col. 6, line 3 to col. 7, line 64; col. 11, lines 10-67	1-16

Further documents are listed in the continuation of Box C. See patent family annex.

* Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A" document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E" earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O" document referring to an oral disclosure, use, exhibition or other means		
"P" document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search

20 JANUARY 1998

Date of mailing of the international search report

10 6 MAR 1998

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT

Washington, D.C. 20231

Facsimile No. (703) 305-3230

Authorized officer

JOSEPH THOMAS

Telephone No. (703) 308-3900

J. Thomas

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/19362

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 4,965,763 A (ZAMORA) 23 October 1990, abstract; figs. 10-18; col. 3, line 19 to col. 7, line 68; col. 30, line 27 to col. 31, line 15; col. 33, line 46 to col. 34, line 16; col. 37, line to col. 38, line 49	1-19
X	FRIEDMAN, C. et al., "Natural Language Processing in an Operational Clinical Information System", Natural Language Engineering, Vol. 1 (1), received 15 August 1994, revised 07 March 1995, copyrighted 1995 and published by Cambridge University Press, pages 83-108, especially pages 83-97 and 104-106	1-16
Y	RUBIN, R. "Can't Reach Your Doctor? Try E-mail.", U.S. News & World Report (magazine article), 13 February 1995, pages 82-83	17-19
A	BENNAHUM, D. "Docs for Docs", Wired (magazine article), March 1995, pages 100, 102, & 104	17-19
A	EL GUEDJ, P.O. et al., "A Chart Parser to Analyze Large Medical Corpora", Proceedings of the 16th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, IEEE, November 1994, pages 1404-1405	1-16

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US97/19362

B. FIELDS SEARCHED

Electronic data bases consulted (Name of data base and where practicable terms used):

APS

search terms: parsing, medical/clinical data, coding/encoding, natural language, domain, filtering